

Towards Wi-Fi AP-Assisted Content Prefetching for On-Demand TV Series: A Reinforcement Learning Approach

Wen Hu, Yichao Jin, Yonggang Wen, *Senior Member, IEEE*, Zhi Wang, *Member, IEEE*,
and Lifeng Sun, *Member, IEEE*

Abstract—The emergence of smart Wi-Fi APs (Access Point), which are equipped with huge storage space, opens a new research area on how to utilize these resources at the edge network to improve users' quality of experience (QoE) (e.g., a short startup delay and smooth playback). One important research interest in this area is content prefetching, which predicts and accurately fetches contents ahead of users' requests to shift the traffic away during peak periods. However, in practice, the different video watching patterns among users, and the varying network connection status lead to the time-varying server load, which eventually makes the content prefetching problem challenging. To understand this challenge, this paper first performs a large-scale measurement study on users' AP connection and TV series watching patterns using real-traces. Then, based on the obtained insights, we formulate the content prefetching problem as a Markov Decision Process (MDP). The objective is to strike a balance between the increased prefetching&storage cost incurred by incorrect prediction and the reduced content download delay because of successful prediction. A learning-based approach is proposed to solve this problem and another three algorithms are adopted as baselines. In particular, first, we investigate the performance lower bound by using a random algorithm, and the upper bound by using an ideal offline approach. Then, we present a heuristic algorithm as another baseline. Finally, we design a reinforcement learning algorithm that is more practical to work in the online manner. Through extensive trace-based experiments, we demonstrate the performance gain of our design. Remarkably, our learning-based algorithm achieves a better precision and hit ratio (e.g., 80%) with about 70% (resp. 50%) cost saving compared to the random (resp. heuristic) algorithm.

Index Terms—Wi-Fi AP, content prefetching, learning-based approach.

I. INTRODUCTION

THESE years have witnessed the explosive growth of network traffic: Cisco [1] had predicted that Global Internet traffic in 2019 will be equivalent to 64 times the volume of the entire global Internet in 2005 and consumer internet video traffic will be 80% of all consumer Internet traffic in 2019, up from 64% in 2014. Although existing video providers have adopted the Content Delivery Network (CDN) to help deliver videos to users across the world, its centralized way and the expensive deployment costs, however, make the conventional CDN not sufficient to provide satisfactory user-perceived QoE [2]. Besides, compared with traditional web objects, the size of video content is several orders of magnitude larger than that of web objects, making the storage space of centralized content servers exhaust much more quickly. This results that the temporal locality of videos can not be well

exploited and users' QoE eventually degrades due to the lower video hit ratio. Moreover, the increasing availability of high quality videos further exacerbate the end users' QoE, e.g., high startup delay and frequent re-buffering events.

To bridge the gap between the explosively increasing network traffic and the slow improvement on the performance of physical network infrastructures, a trend has emerged to shift the traffic at peak periods by prefetching them in advance. Some works [3] have proposed to utilize the *edge-devices* (e.g., set-top boxes, broadband gateway) to assist the video delivery. This approach becomes more promising thanks to the emergence of smart APs. Compared with tradition APs which only perform the data forwarding function, these smart APs (also called home router) are equipped with an OS and some storage devices (e.g., a hard disk drive or SD card). Furthermore, the enormous popularity of smart AP (6.5 M sales until 2015 in China [4]) enables these widely distributed resources to conduct the prefetching tasks.

However, in practice, the different video watching patterns among users, and the varying network connection status lead to the time-varying server load, which eventually makes prefetching efficiently challenging, i.e., we need to address the following challenges: i) What content should be prefetched to which AP ? ii) How many content items should be prefetched ? Intuitively, there is a trade-off between the prefetching costs and users' QoE, in the AP-assisted content prefetching problem. On one hand, aggressive prefetching strategy can guarantee a higher probability of hit ratio, thus a better users' QoE. On the other hand, more contents to prefetch, in turn, involves more monetary cost. Moreover, prefetching aggressively will incur the competition for the end user's downlink bandwidth with the current playback task, which will degrade the current viewing experience.

To address these challenges, we perform a large-scale measurement study on users' AP connection and TV series watching patterns using real-traces. Based on the obtained insights, we formulate the content prefetching problem as a Markov Decision Process (MDP) and propose four algorithms to solve it.

Our contributions in this paper are summarized as follows:

▷ We carry out large-scale measurement study on 270 M user-AP association traces and 1.8 M users' watching traces on 76 K episodes from 8.5 K TV series. The observations on users' AP connection and their TV series watching patterns are reported as follows: i) The user's connected APs are stable

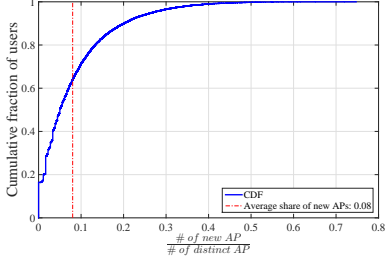


Fig. 1. The fraction of the users vs. the ratio between new APs and distinct APs associated by each user per day.

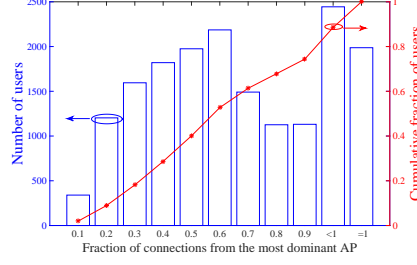


Fig. 2. The fraction of users vs. the contribution of each user's the most dominant AP.

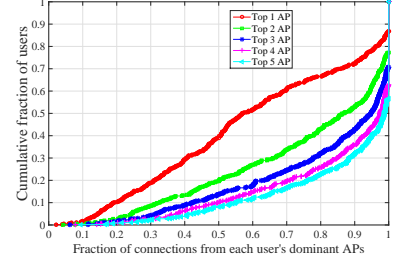


Fig. 3. The cumulative fraction of users vs. the contribution of each user's top- K dominant APs.

over time; ii) There are more than 60% (*resp.* 90%) users with the majority (e.g., over 50%) of their connections served by their top 1 (*resp.* 5) AP(s); iii) Users tend to watch consecutive episodes in the same TV series. These observations motivate our work and provide valuable insights for our ensuing design.

▷ Based on our measurement studies, we propose the AP-assisted content prefetching paradigm and mathematically formulate the content prefetching problem as a Markov Decision Process. The objective is to balance the trade-off between various costs (including transmission, storage, increased latency and resource competition cost) and users' QoE.

▷ Under this framework, we propose four algorithms to schedule the content prefetching tasks in different periods. First, we obtain the lower and upper performance bound by proposing a random fixed and an offline algorithm, respectively. Then, we present a heuristic algorithm as another practical baseline. Finally, we propose a reinforcement learning algorithm to learn the history traces and schedule content prefetching accordingly in the online manner.

▷ Using trace-driven experiments, we further evaluate the performance of each algorithm. The results show that our design is adaptive to the server load and achieves about 70% (*resp.* 50%) cost saving over the random (*resp.* heuristic) algorithm with high precision and hit ratios.

The rest of the paper is organized as follows. We present the measurement insights that motivate our design in Sec. II. We present the system architecture and the problem formulation in Sec. III. We propose our strategies in Sec. IV. We evaluate their performance in Sec. V. We discuss the related works in Sec. VI. Finally, we conclude this work in Sec. VII.

II. MEASUREMENT AND ANALYSIS

In this section, we first introduce the datasets utilized in this paper. Then, we present some insights we learned from the exhaustive measurement study.

A. Dataset

1) *Traces of AP Connections*: We study users' AP association patterns using the dataset provided by Tencent¹. The dataset contains 270 M user-AP association traces during one month (March 2015 - April 2015). Each trace item records the

user ID, the Basic Service Set Identifier (BSSID) of the AP, the timestamp of the association, and the location of the AP. Note that the user ID and the BSSID are unique for different users and APs, respectively. Thus we can identify each specific user and AP to learn the users' AP connection patterns.

2) *Traces of TV Series Sessions*: We investigate users' TV watching behavior, i.e., the transition among episodes in the same TV series, based on the traces provided by iQiyi², one of the most popular online video providers in China. The traces are collected from 1.8 M users in a metropolitan city during 2 weeks of May 2015, containing 76 K episodes on 8.5 K TV series. In particular, the traces are recorded at the request level, i.e., the watching experience for an episode is recorded as one session. In each trace item, the following information is recorded: the user ID, the timestamp when the user starts to watch the video and the title of the episode. Based on these traces, we will study the users' TV watching behavior.

B. User-AP Connection Pattern

First of all, we calculate the ratio between the number of new APs, which have not been connected previously, and that of distinct APs associated by each user per day. We plot the cumulative distribution of aforementioned ratio in Fig. 1. It is clearly that the number of new APs is much smaller than that of distinct APs. In particular, about 80% users visit less than 20% new APs per day and the average share of new APs is only 8%.

Next, we explore the users' preference to a AP, which is defined as the ratio between the connections served by one AP and the total connections issued by a user, in Fig. 2. In the analysis, we focus on the users who access network via APs at least once a day, and filter others. The bar represents the number of users whose preference to a AP falls in the corresponding ratio bins (the length of each bin is 0.1) in the x axis. We observe that there are about 2500 users (the highest bar) with more than 90% connections served by his most dominant AP. From the cumulative distribution, we observe that about 60% users with 50% connections served by his most dominant AP. Furthermore, we investigate how much each user's top- K APs (ranked by the number of connections) contribute for his total requests in Fig. 3. We observe that there

¹<http://www.tencent.com>. Hereinafter, The Wi-Fi Service Provider is referred to the service.

²<http://www.iqiyi.com>. Hereinafter, The Video Service Provider is referred to the service.

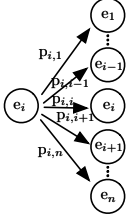


Fig. 4. Possible transition among episodes in the same TV series.

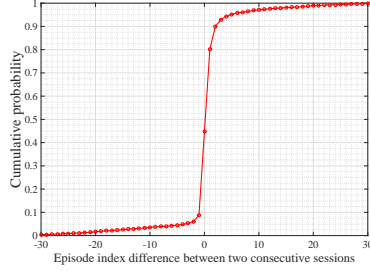


Fig. 5. Distribution of episode index difference between two consecutive sessions for a demonstrative TV series consisting of 33 episodes.

are more than 60% (*resp.* 90%) users with the majority (e.g., over 50%) of their connections served by their top 1 (*resp.* 5) AP(s).

Given these observations, we claim that the AP set connected by each user shows an obvious stability, i.e., each user associates with familiar APs frequently and a few APs (e.g., the top 1 AP) serve the majority, if not all, of the user's requests. In this paper, we focus on the case study for *one user and one AP*, where each dominant AP only serves for one user and the prefetching strategies proposed later are only conduct on each user's dominant AP, for the following reasons: i) The contribution of the most dominant AP is significant. ii) In our dataset, less than 9% users have the same dominant AP with others. Presumably, each user's most dominant AP is his home AP. We will investigate the collaborative content prefetching among multiple APs in our future works.

C. User TV Series Watching Pattern

As illustrated in Fig. 4, after viewing the current episode, the user is likely to watch every episode of a TV series in consecutive time slots. By studying the video traces, we plot the transition probability distribution of the episodes index difference between two consecutive sessions in a demonstrative TV series in Fig. 5. We observe that users are prone to either keep watching the current episode or the next three episodes, with a probability of 35%, 47% respectively. The rationale is that there is strong relationship among the plot development of adjacent TV series. Besides, we observe that there is less possibility (about 18%) for large forward/backward episode transitions (large episode transitions are defined as the index differences range from -30 to -1 or from 4 to 30) and the cumulative possibility increases linearly with the large index differences. This is reasonable since the large episode transition indicates that the user does not care the plot development, as such, each episode is considered equally and the possibility for jumping to each episode is uniform.

These results motivate us to design the AP-assisted strategy to improve users' quality of experience (QoE) by prefetching more content while accounting for the fact that more prefetched content incur more additional monetary cost and resource competition costs. Details on the various cost models will be elaborated in the ensuing section (Sec. III-B4).

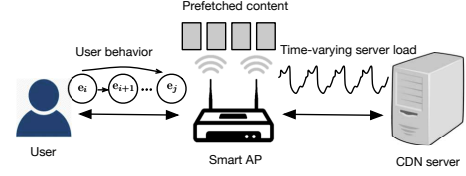


Fig. 6. The architecture of AP-assisted content prefetching.

III. MODEL & FORMULATION

In this section, we present some models based on the MDP framework [5] and formulate the AP-assisted content prefetching problem as an unconstrained optimization problem.

A. System Architecture

We illustrate the proposed system architecture in Fig. 6. The CDN server is the source of content and the server load is dynamic over time. Since the smart APs (e.g., HiWiFi³, MiWiFi⁴, Newifi⁵) are equipped with huge storage space, they are potential to help the content delivery. In this system, the distributed APs and the CDN servers are rent by the content service provider. Given the observation of time-varying server load, the content service provider should adopt the prefetching technique to shift traffic at peak periods, i.e., content should be prefetched to APs at server idle time slots. As such, when a request arrives, the AP first checks whether this request can be satisfied locally, i.e., whether the requested content has been stored in the AP. Otherwise, the request will be redirected to the remote CDN server, thereby incurring a higher delay. Meanwhile, the decision on which videos should be prefetched is driven by the users' watching behavior, i.e., the transition between episodes, to improve the possibility that the prefetched videos will be actually watched in the future.

B. System Assumptions

1) *Content Model*: In this paper, we assume that the content catalog does not change in some periods (e.g., several hours or days). Especially, we focus on the on-demand TV series and assume the size of each TV episode is the same. Since the Time-to-Live (TTL) based caching policy: i) decouples the eviction mechanism among content [6]; ii) captures the properties of existing popular eviction policies [7] (e.g., LRU, FIFO and RND), we assume that the content stored in AP should be evicted after a time threshold T_{th} is due, i.e., each content has a lifetime. Given the observation that the mean watching finish ratio of each episode is 72% [8], we treat an episode as an unit and do not consider the partial prefetching. Furthermore, due to the facts: i) The end user's downlink bandwidth is limited; ii) Prefetching too many content items will lead to content eviction before being watching, we assume that at most K_{th} videos can be prefetched during one time slot.

³<http://www.hiwifi.com>.

⁴<http://www.miwifi.com>.

⁵<http://www.newifi.com>.

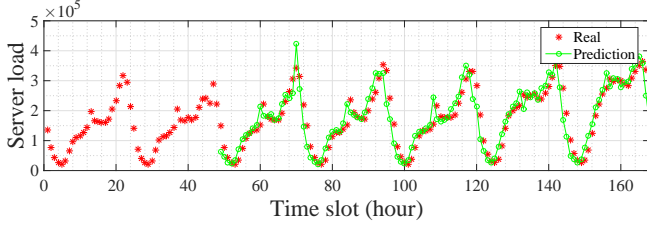


Fig. 7. CDN server load over time measured using traces from The Video Service Provider.

2) *User Behavior Model*: We model users' transition behavior among series in the same TV series as a Markov model based on the insights learned from the measurement study in Sec. II-C. As opposed to the short videos published on social video sharing sites where users browser videos quickly [9], users tend to complete watching episodes of TV series [8]. We set the duration of each time slot the same value for each episode, thus the user only watches one video in one time slot. Since the prediction of periods when users will watch videos is outside the scope of the paper, our design works in a conservative way, i.e., our design only works in *effective time slots*, which are defined as the time slots when a user is actually watching videos and we do not consider content prefetching during time slots when he does not consume videos.

3) *Network Traffic Model*: We study the number of video requests hourly from the same dataset as we used in Sec. 2, to build the network traffic model. From Fig. 7, we observe the marked daily pattern of the video request traces, which motivates us to adopt the seasonal ARIMA model (Autoregressive Integrated Moving Average) [10] to predict the server load. Mathematically,

$$\Phi(L^s)\phi(L)\nabla_s^D\nabla^dX_t = \theta(L)\Theta(L^s)\varepsilon_t, \quad (1)$$

where Φ (resp. ϕ) and Θ (resp. θ) are the seasonal (resp. non-seasonal) autoregressive and moving average parts. ε_t is the white noise for the stationary distribution. L is the lag operation, i.e., $L^dX_t = X_{t-d}$. ∇ is the difference operator, i.e., $\nabla X_t = X_t - X_{t-1}$ and thus, $\nabla = 1 - L$. D (resp. d) is the the order of seasonal (resp. non-seasonal) integration part and s denotes the number of periods in a season. Due to the limit of space, we omit the details of ARIMA model which can be found in [10].

By training with the historical traces, the model is of the form of $(0, 1, 1) \times (0, 1, 1)_{24}$ and there are 24 periods, one for every hour of the day, in a season. The prediction results are presented in Fig. 7, each green dot is predicted by learning the server load in previous 48 hours. We observe that the prediction achieves a relatively accurate estimation (e.g., Mean Absolute Percent Error is 17.52%) of the server load with a small learning window.

4) *Cost Model*: The costs for content prefetching consist of several parts, including: the transmission cost for prefetching the content from the server to the AP or directly downloading to the end user; the storage cost for holding the content in the AP; the QoE degradation cost for the increased delay

by fetching the content from the server rather than the AP and the resource competition cost for the higher startup delay incurred by limited resource competition. In particular, the QoE degradation cost and resource competition cost are designed to reward the system for the improvement of user-perceived quality of experience.

Transmission Cost: By taking the server load, which can be predicted with the network traffic model in Sec. III-B3, into consideration, we adopt a server load aware transmission cost function $\Psi(l)$. It is worth noting that there is no requirement for the exact definition of $\Psi(l)$ as long as it is a increasing convex function. Here, we follow the work in [11] and adopt the logarithmic barrier function as follows.

$$\Psi(l) = -\log(1 - \frac{l}{l_{th}}), \quad (2)$$

where l is the current server load and l_{th} is constant variable which denotes the server threshold load. Therefore, $\Psi(l)$ is a strictly increasing convex function with respect to the server load. The rationale behind $\Psi(l)$ is that it is cheap to prefetch content when the server is under small utilization whereas when the load approaches the threshold load l_{th} , we get highly penalized to guarantee that content prefetching should never happen.

Thereby, the cost for content transmission is defined as follows.

$$C^{tr}(x) = \beta\Psi(l)x, \quad (3)$$

where x is the number of content items to be downloaded/prefetched from the CDN server and β is the tuning parameter to guarantee that the median of the transmission cost per content is consistent with the Amazon on-demand pricing model [12] and the details will be elaborated in Sec. V-A1.

Storage Cost: The cost for storing content in AP for one time slot is defined as follows.

$$C^{st}(x) = \kappa x, \quad (4)$$

where κ is the fixed cost per content per time slot in the AP, x is the number of stored content items.

Latency Increase Cost: We also consider the cost introduced by the QoE degradation due to cache miss on AP, as follows.

$$C^{la}(x) = (d^1 - d^0)x, \quad (5)$$

where x is the number of content items to be downloaded from the CDN server, d^0 is a fixed variable denoting the startup delay for downloading the content from the AP, while d^1 is a volatile variable denoting the startup delay for downloading the content from the server and its value depends on the current server load.

Resource Competition Cost: Since we schedule content prefetching while the user is watching a video, the prefetching task competes for the limited resources, e.g., the end users' downlink bandwidth, with the downloading task for the current playback. The competition incurs that the assigned bandwidth for the playback task decreases. Based on the relationship

between QoE and the bandwidth in [13], we define the competition cost for content prefetching as follows.

$$C^p(x, y) = \begin{cases} 0 & x = 0; \\ 0 & y = 0; \\ \log(bw) - \log(\frac{bw}{x+y}) = \log(x+y) & \text{otherwise,} \end{cases} \quad (6)$$

where bw is the residential downlink bandwidth, x is the number of content items to be prefetched in parallel with the video playback and y is a binary variable, i.e., $y = 0$ if the being watched video has been previously prefetched; otherwise, $y = 1$. It is reasonable to expect that the resource competition cost is zero either when there are no prefetching tasks or the video being watched has been previously prefetched. Note that, for simplicity, we consider that the bandwidth is allocated among the tasks equally.

C. Problem Formulation

1) *System States*: Let $\mathcal{S} = \{s_1, s_2, \dots, s_t, \dots, s_T\}$ denote the state space. $s_t = (e_t^w, \mathbf{E}_t^{st})$, where e_t^w is the content actually being watched by the user at time slot t and \mathbf{E}_t^{st} is the content set whose elements have been stored in the AP. Therefore, each state can represent both the episode user is watching and the content set stored in the AP.

2) *Actions*: Let $\mathcal{A} = \{a_1, a_2, \dots, a_t, \dots, a_T\}$ denote the action set adopted by the system over time. $a_t = (\mathbf{E}_t^{tr}, \mathbf{E}_t^d)$, where \mathbf{E}_t^{tr} and \mathbf{E}_t^d are the content set scheduled to be prefetched and deleted at time slot t , respectively. Note that no action is taken for prefetching if the size of \mathbf{E}_t^{tr} is 0 and the elements of \mathbf{E}_t^d are determined by the content lifetime based on the eviction policy presented in Sec. III-B1.

3) *System State Transition*: Intuitively, the system state in time slot $t+1$ is $s_{t+1} = (e_{t+1}^w, \mathbf{E}_{t+1}^{st})$, where \mathbf{E}_{t+1}^{st} can be calculated based on its previous state at time slot t , as follows.

$$\mathbf{E}_{t+1}^{st} = \mathbf{E}_t^{st} \cup \mathbf{E}_t^{tr} \setminus \mathbf{E}_t^d. \quad (7)$$

The transition possibility from s_t to s_{t+1} can be calculated as follows.

$$\begin{aligned} P_{a_t}(s_t, s_{t+1}) &= P(s_{t+1} | s_t, a_t) \\ &= P((e_{t+1}^w, \mathbf{E}_{t+1}^{st}) | (e_t^w, \mathbf{E}_t^{st}), a_t) \\ &= P(e_{t+1}^w | e_t^w) P(\mathbf{E}_{t+1}^{st} | \mathbf{E}_t^{st}, a_t). \end{aligned} \quad (8)$$

The last equality in Eq. (8) follows because the possibility of episodes' transition is independent of the action.

4) *Cost Function*: The cost function at time slot t is defined as the weighted sum of the aforementioned parts of costs (Eq. 9), including the monetary-related cost part (Eq. 10) and the QoE-related cost part (Eq. 11).

$$g_t(s_t, a_t) = C^m + \lambda_1 C^q, \quad (9)$$

$$C^m = C^{tr}(\|\mathbf{E}_t^{tr}\|) + C^{tr}(\delta(e_t^w, \mathbf{E}_t^{st})) + C^{st}(\|\mathbf{E}_t^{st} \cup \mathbf{E}_t^{tr}\|), \quad (10)$$

$$C^q = C^{la}(\delta(e_t^w, \mathbf{E}_t^{st})) + \lambda_2 C^p(\|\mathbf{E}_t^{tr}\|, \delta(e_t^w, \mathbf{E}_t^{st})), \quad (11)$$

where λ_1 is a positive trade-off parameter to balance the trade-off between the C^m and C^q , λ_2 is a positive tuning parameter for balancing the relationship between latency cost C^{la} and

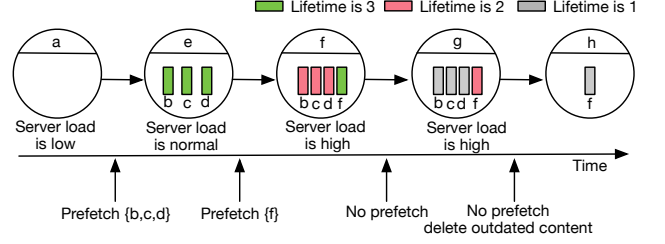


Fig. 8. Illustration of the content prefetching according to the user behavior and the time-varying server load.

resource competition cost C^p , $\|\cdot\|$ is the operator of calculating the number of elements in the argument set “.” and $\delta(e_t^w, \mathbf{E}_t^{st})$ is an indicator function defined as follows.

$$\delta(e_t^w, \mathbf{E}_t^{st}) = \begin{cases} 0 & e_t^w \in \mathbf{E}_t^{st}; \\ 1 & \text{otherwise.} \end{cases} \quad (12)$$

Our objective is to minimize the total cost over a finite time horizon, i.e.,

$$\min_{\{a_t\}} \sum_{t=0}^{T-1} g_t(s_t, a_t). \quad (13)$$

IV. STRATEGIES FOR AP-ASSISTED CONTENT PREFETCH

In this section, we first present an illustrative example to elaborate details of the problem. Then we start with some naive algorithms, including a random fixed algorithm for the lower performance bound, and a backward induction algorithm for the upper performance bound. Besides, we also propose a heuristic algorithm as another baseline. Finally, we adopt the reinforcement learning algorithm to design an online strategy, which is more practical.

We illustrate the main idea behind our design by a simple example in Fig. 8. In this case, the users' video watching sequence is $a \rightarrow e \rightarrow f \rightarrow g \rightarrow h$ and the T_{th} is 3, i.e., a content should be evicted after 3 time slots. Since the server load is low when user is watching video a , it is optimal to prefetch many content based on the transition probability. Less or no content should be prefetched when the server is overloaded. Note that in the last state transition, content b, c, d are evicted because of time out. Intuitively, the optimization objective is to make sequential optimal decisions on the content prefetching over a finite time horizon.

A. Random Algorithm & Performance Lower Bound

In order to obtain the lower performance bound, we consider the naive strategy which prefetches a fixed number of content and the content are randomly selected. Since the system has no idea of the user behavior and the server load information in the future, how many and what content should be prefetched are hard to decide. As such, all the choices within the prefetch threshold K_{th} , i.e., no prefetching, one episode prefetching, and so forth, are enumerated and a fixed number of content are randomly selected from the episodes set of a TV series. As for each choice, the total cost is calculated as the sum of cost

at each timeslot (Eq. (9)). Recall that the cost at each timeslot is calculated as the weighted sum of the monetary-related cost and the QoE-related cost. After the traversal, the average cost of all choices is calculated as the cost expectation.

B. Heuristic Algorithm

We also propose a heuristic algorithm for performance comparison. As opposed to the random algorithm, the heuristic algorithm incorporates the users' TV watching pattern studied in Sec. II-C into the strategy design. Since users are prone to either keep watching the current episode or the next three episodes, we choose to prefetch content from the next three episodes. However, the next period when the user will watch videos is unknown beforehand, thus we can not decide the optimal number of content to be prefetched based on the varying server loads at the current period and next period. In this algorithm, we enumerate the possible choices on the number of prefetching and calculate the average cost of all choices as the cost expectation.

C. Offline Algorithm & Performance Upper Bound

In order to obtain the upper performance bound, we assume the complete user behavior trace $\mathcal{H} = \{e_1^w, \dots, e_T^w\}$ is known in prior, which means that not only the information before current time but also the future information, e.g., what and when the user will watch, are available. Thus, we can design the offline algorithm which achieves the optimal performance. Given this assumption, the original problem reduces to a deterministic MDP and the transition possibility can be calculated as follows.

$$\tilde{P}_{a_t}(s, s') = \begin{cases} 1 & e^w = e_t^w, e^{w'} = e_{t+1}^w, \mathbf{E}^{st'} = \mathbf{E}_t^{st} \cup \mathbf{E}_t^{tr} \setminus \mathbf{E}_t^d; \\ 0 & \text{otherwise.} \end{cases} \quad (14)$$

Since the Bellman optimality equation [14] characterizes the value function and calculates the optimal value in state s as the sum of the immediate cost and the discounted optimal value in the next state, we define the value function of taking action a_t in state s_t as $\mathbf{V}_t(s_t, a_t)$, which represents the expected accumulative cost from the current time slot t to the end T .

$$\mathbf{V}_t(s_t, a_t) = g_t(s_t, a_t) + \sum_{s_{t+1} \in \mathcal{S}} \tilde{P}_{a_t}(s_t, s_{t+1}) \mathbf{V}_{t+1}^*(s_{t+1}). \quad (15)$$

The minimal value function $\mathbf{V}_t^*(s_t)$ can be derived as follows. Note that we initially set $\mathbf{V}_T^*(s_T) = 0$ for all $s_T \in \mathcal{S}$.

$$\mathbf{V}_t^*(s_t) = \min_{a_t \in \mathcal{A}} \mathbf{V}_t(s_t, a_t). \quad (16)$$

After finding the optimal value of $\mathbf{V}_t(s_t, a_t)$, we can derive the optimal policy using the following equation.

$$\pi(s_t, t) = \arg \min_{a_t \in \mathcal{A}} \mathbf{V}_t(s_t, a_t), \quad (17)$$

where π is the policy which maps the state and stage to actions, i.e., $\pi : s \times t \rightarrow a$.

The details of this method are presented in Algorithm 1. It adopts the value iteration technique, which utilizes the Bellman optimality equation, to iteratively compute the expected

Algorithm 1: Offline Algorithm for MDP.

Input : a complete user behavior traces \mathcal{H} for a TV series

Output: optimal policy $\pi(s_t, t)$ at each time slot

```

1 initialize the value function  $\mathbf{V}_T^*(s_T) = 0$  for all
    $s_T \in \mathcal{S}$ 
2 for  $t = T - 1, \dots, 1$  do
3   for each  $a_t \in \mathcal{A}$  do
4     calculate the transition possibility  $\tilde{P}_{a_t}(s_t, s')$ 
       with Eq. (14)
5     update the value function  $\mathbf{V}_t(s_t, a_t)$  with
       Eq. (15)
6   end
7   derive the minimal value function  $\mathbf{V}_t^*(s_t, a_t)$  with
       Eq. (16)
8 end
9 for  $t = 1, \dots, T - 1$  do
10  derive the optimal policy  $\pi(s_t, t)$  with Eq. (17)
11 end
```

minimal accumulated cost at each time slot. As such, the optimal policy at each time slot can be derived. In particular, the value iteration begins at the end time slot T with an initial value, e.g., 0, for the value function $\mathbf{V}_T^*(s_T)$ (line 1) and works backward, calculating the value at each time slot (line 2–line 8). After that, the optimal policy is derived forward based on the stored values (line 9–line 11).

D. Online Algorithm with Approximate Reinforcement Learning

In this part, we propose an online algorithm using reinforcement learning. Firstly, the problem of the large state and action spaces for our content prefetching paradigm is analyzed. Then, we propose to use the function approximation technique, i.e., the use of a parameterized functional form to represent the value function.

1) *Curse of Dimensionality*: Based on the definition of state and action variable in Sec. III-C, we analyze the dimensionality of state space and the action space. Recall that $s_t = (e_t^w, \mathbf{E}_t^{st})$ consists of two parts: the current watching episode and the buffered content set. As for a TV-series with m episodes and each prefetched content with T_{th} lifetime, there are $\|\mathcal{S}\| = m(T_{th} + 1)^m$ states. Similarly, the action variable $a_t = (\mathbf{E}_t^{tr}, \mathbf{E}_t^d)$ consists two parts: the content to be prefetched and the content to be deleted. The number of content to be prefetched is smaller than the threshold K_{th} and the deletion action is enforced by the lifetime threshold (T_{th} time slots). Hence, there are $\|\mathcal{A}\| = \binom{m}{1} + \binom{m}{2} + \dots + \binom{m}{K_{th}}$ actions for each state, where the first term indicates a random prefetching on just one episode, the second term indicates a random prefetching on two episodes, and so forth. As for our problem, the classic *tabular* method, which represents the value function as a table with an entry for each state or state-action pair, could not be adopted. Because the large size of the table ($\|\mathcal{S}\| \times \|\mathcal{A}\|$) requires many memory and much time to accurately calculate them.

2) *Gradient-based Q-learning*: Approximation-based value function for problems with large space is an active research topic on reinforcement learning [15], which adopts approximate functions to generalize the value of states. And we will reduce the state and action dimensionality by feature extraction and a heuristic policy, respectively.

Compact Parameterized Function Representation: As for the state space, we adopt a feature-extraction function $\phi: \mathcal{S} \rightarrow \Phi$ to map states into features in the feature space Φ . Corresponding to each state s , there is a feature vector $\mathbf{x}(s)$, $\phi(s) = \mathbf{x}(s) = (x_1(s), x_2(s), \dots, x_n(s))^T$. In our problem, we represent an episode with a $(T_{th} + 2)$ binary vector: i) The first component is 1 if the episode is being watched; otherwise, it is 0. ii) The $(i + 1)$ -th component of the binary vector is 1 if the remaining lifetime of the episode is i ; otherwise, it is 0. Hence, each state can be represented with a binary vector of size $m(T_{th} + 2)$. After feature extraction, the approximated Q function is defined as follows.

$$Q_t(s_t, a_t) = \theta_t(a_t)\phi(s_t) = \theta_t(a_t)\mathbf{x}(s_t), \quad (18)$$

where θ_t denotes the adaptable parameter matrix of size $\|\mathcal{A}'\| \times \|\mathcal{S}'\|$ at time t and $\theta_t(a_t)$ denotes a_t -th row of θ_t . \mathcal{A}' and \mathcal{S}' denotes the reduced action space and state space, respectively.

The dimensionality of the action space is reduced by the insights learned in Sec. II-C. Since the probability that user jumps to the $(i + 4)$ -th episodes after watching i -th is too small, we only consider three classes of actions: prefetch one, two or three episodes from next consecutive three episodes. And these classes have 3, 3, 1 choices, respectively.

By representing in the compact way, at each time slot, we only need to learn the parameter matrix θ_t . In our problem, $\|\mathcal{A}'\| = 7$, $\|\mathcal{S}'\| = m(T_{th} + 2)$, and the size of θ_t is $\|\mathcal{A}'\| \times \|\mathcal{S}'\| = 7 \times m(T_{th} + 2)$, which is very manageable.

To summarize, we dramatically reduce the dimensionality from $\|\mathcal{A}\| \times \|\mathcal{S}\|$ to $\|\mathcal{A}'\| \times \|\mathcal{S}'\|$.

Gradient-descent Update: We quantify the one step temporal-difference error as $f(s_t, a_t) = \frac{1}{2}(\delta_t)^2$, where $\delta_t = \gamma \min_a Q_t(s_{t+1}, a) + g_t(s_t, a_t) - Q_t(s_t, a_t)$, and γ ($0 \leq \gamma \leq 1$) is the discount factor, which can be interpreted intuitively as a way of trading off the importance of sooner and later cost.

In order to minimize the temporal-difference error, we first calculate the negative gradient of $f(s_t, a_t)$ as follows.

$$\begin{aligned} -\nabla_{\theta_t} f(s_t, a_t) &= -\delta_t \nabla_{\theta_t} (\delta_t) \\ &= \delta_t \nabla_{\theta_t} Q_t(s_t, a_t). \end{aligned} \quad (19)$$

Then, we update the parameters with Q-learning along the negative gradient descent direction, in which the error falls most rapidly, as follows.

$$\begin{aligned} \theta_{t+1}(a_t) &= \theta_t(a_t) - \alpha \nabla_{\theta_t} f(s_t, a_t) \\ &= \theta_t(a_t) + \alpha \delta_t \nabla_{\theta_t} Q_t(s_t, a_t) \\ &= \theta_t(a_t) + \alpha \delta_t \phi(s_t), \end{aligned} \quad (20)$$

where α_t ($0 < \alpha \leq 1$) is a step-size parameter, which influences the step learning rate, i.e., how much the newly acquired information will override the old information.

Algorithm 2: Q-learning with Experience Replay.

Input : discount factor γ , learning rate α , feature-extraction function ϕ , greedy- ϵ policy, user behavior history $\mathcal{H} = \{e_1^w, \dots, e_{T'}^w\}$ and the convergence threshold χ

Output: the parameter matrix θ

```

1 initialize the parameter matrix  $\theta_1$  (e.g., 0)
2 for  $t = 1, \dots, T' - 1$  do
3   map  $s_t$  to a feature vector  $\mathbf{x}(s) = \phi(s)$ 
4   apply  $a_t = \pi(s_t, t)$  with Eq. (21)
5   measure the next state  $s_{t+1}$  and the cost  $g_t(s_t, a_t)$ 
6   update the parameter  $\theta_t$  with Eq. (20)
7    $\epsilon = \epsilon - \epsilon'$ 
8 end
9 if any  $|\theta_t - \theta'_t| \geq \chi$  then
10  collect more user behavior history
11 else
12  return parameter matrix  $\theta$ 
13 end
```

Exploration-exploitation Balance: In order to guarantee that the Q-learning converges to the optimal Q-function, we adopt the ϵ -greedy policy to balance the exploration of selecting any action with a non-zero probability with exploitation of selecting the greedy actions in the current Q-function. In particular, the action is selected according to

$$\pi(s_t, t) = \begin{cases} \arg \min_{a \in \mathcal{A}'} (\theta_t(a)\mathbf{x}(s_t)) & \text{with probability } (1 - \epsilon); \\ \text{random action from } \mathcal{A}' & \text{with probability } \epsilon, \end{cases} \quad (21)$$

where ϵ ($0 \leq \epsilon < 1$) is the exploration probability at state s_t and ϵ diminishes over time with a rate ϵ' .

Algorithm Design: In order to speed up the learning process, we extend the Q-learning with the experience replay [16]. The details of the online approach are presented in Algorithm 2 and Algorithm 3.

During the experience replay process (Algorithm 2), we first extract the feature vectors $\mathbf{x}(s_t)$ for each state (line 3) and select the action based on the ϵ -greedy policy. After performing the action, the parameter matrix θ_t is updated based on the one step temporal-difference error (line 6). This process continues until the parameter matrix θ_t begins to converge. Otherwise, more traces will be collected for the experience replay. The replay process is guaranteed to converge when the MDP problem is finite [17].

After obtaining the converged parameter matrix θ_t at each time slot, we can derive the Q-value for any state at the current time slot. As illustrated in Algorithm 3, the policy is derived by choosing the action with minimum Q-value $Q_t(s_t, a_t)$ (line 6). Note that the learning process continues for a better policy derivation in the future (line 8).

V. PERFORMANCE EVALUATION

In this section, we numerically evaluate the effectiveness and performance of our AP-assisted content prefetching framework based on trace-driven simulations. Specifically, we use

Algorithm 3: Online Learning with Function Approximation.

Input : discount factor γ , learning rate α ,
feature-extraction function ϕ , state s_t at time
slot t , and the learned parameter matrix θ_t at
each time slot from experience replay process

Output: optimal policy $\pi(s_t, t)$ at each time slot

```

1 for  $t = 1, \dots, T - 1$  do
2   map  $s_t$  to a feature vector  $\mathbf{x}(s) = \phi(s)$ 
3   for each  $a_t \in \mathcal{A}'$  do
4     update the approximated  $Q$  function
        $Q_t(s_t, a_t)$  with Eq. (18)
5   end
6   apply  $\pi(s_t, t) = \arg \min_{a \in \mathcal{A}'} Q_t(s_t, a)$ 
7   measure the next state  $s_{t+1}$  and the cost  $g_t(s_t, a)$ 
8   update the parameter  $\theta_t$  with Eq. (20)
9 end

```

the real traces from The Video Service Provider to characterize users' watching patterns.

A. Experiment Setup

1) *Parameters Setting:* According to the Amazon's on-demand model [12], i.e., the storage cost is 2×10^{-4} USD/GB per hour and the transmission cost is 0.12 USD/GB, we calculate the corresponding cost parameters in our cost model as follows: i) The storage cost per content item per time slot κ equals 6×10^{-5} . Note that we consider that all the episodes are high definition (HD videos), i.e., 720p video settings, and the duration of each episode is 45 minutes, thus the size of each episode is 400 MB. ii) We adjust the tuning parameter β as 0.16 in the cost model to guarantee the median of transmission cost for a content is 0.048 USD, which is consistent with the pricing model [12] widely adopted by the literature [18]. iii) Since users start to abandon the video if the startup delay exceeds about 2 s [19], the video startup delay from the CDN server d^1 is less than 2 s and is proportional to the degree of the server load. The video startup delay from AP d^0 equals 0.05 s. iv) For Q-learning settings, we set the learning rate $\alpha = 0.5$ to give equal weight to new and old knowledge, the discount factor $\gamma = 0.99$ to take more future costs into account, and the Q-value convergence threshold $\chi = 0.0001$. We initially set $\epsilon = 0.5$ and the reduction rate $\epsilon' = 0.0005$ per time slot.

Since content prefetching will incur competition cost at the current time slot and may reduce the startup delay in the next time slot, the tuning parameter λ_2 for balancing the relationship between latency cost C^t and resource competition cost C^p must be less than 1. Otherwise, it will never be optimal to prefetch any content, because the benefit of content prefetching, i.e., the reduced startup delay, will be offset by the incurred competition cost. In the following experiments, if not otherwise specified, λ_2 is 0.02 and the trade-off parameter between the monetary-related cost and the QoE-related cost

TABLE I
SIMULATION PARAMETERS SETTING.

Parameter	Value
Lifetime threshold T_{th}	3 time slots
Prefetch content threshold at each time slot	3
K_{th}	
Duration of one time slot	45 min
Storage cost per content item per time slot κ	6×10^{-5} \$
Tuning parameter for transmission cost per content item β	0.16
Startup delay from CDN server d^1	$2 \times \frac{l}{l_{th}}$ s
Startup delay from AP d^0	0.05 s
Discount factor of online learning γ	0.99
Learning rate of online learning α	0.5
Convergence threshold χ	0.0001

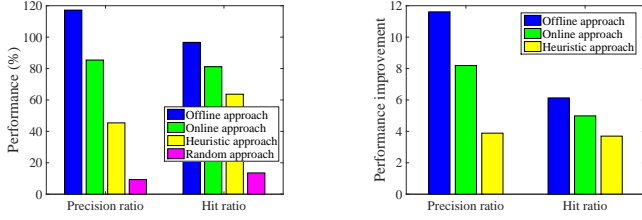
(λ_1) is 0.9. The details of experiment setting are summarized in Table I.

2) *Trace-driven Simulation:* According to the video session traces from The Video Service Provider, we simulate each user's behavior based on the request patterns and simulate the server load based on the overall requests from users at each time slot. Since the collected traces spanning limited periods, i.e., 2 weeks, we focus on the TV series with 30 episodes and remove users who do not issue more than 30 requests for any TV series. It is worth noting that because the user may keep watching one episode in several consecutive time slots, 30 requests for a TV series do not indicate the user watches 30 distinct episodes. The performance of each algorithm presented in the following section is the average results of 1000 rounds.

3) *Metrics:* In order to quantify the performance of different prefetching strategies, we adopt the following metrics: i) Precision ratio (PR) [20]: the ratio between the number of the useful prefetches p_u and the total number of prefetched videos p_t . Mathematically, $PR = \frac{p_u}{p_t}$. Here, p_u is defined as the number of prefetched videos which are actually consumed before eviction. ii) Hit ratio (HR) [20]: the ratio between the useful prefetches p_u and the total number of requested videos r_t . Mathematically, $HR = \frac{p_u}{r_t}$. iii) Various costs formulated in Sec. III-C4, including the overall cost $g_t(s_t, a_t)$, the monetary-related cost C^m and the QoE-related cost C^q .

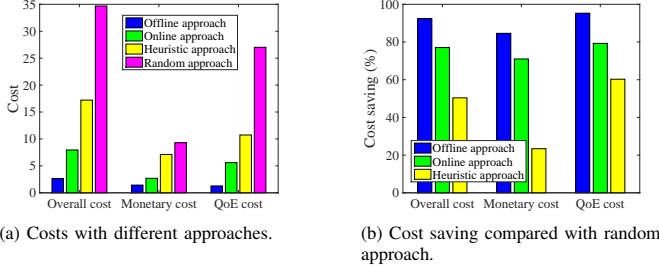
B. Experiment Results

1) *Effectiveness of Our Proposed Strategies:* In this part, we evaluate the effectiveness of our proposed online approach by comparing it with baselines. First, we compare these algorithms in terms of the precision ratio and the hit ratio. As shown in Fig. 9, we make the following observations: i) The online approach can achieve about 80% precision accuracy, which is 9 (resp. 1.8) times better than the random (resp. heuristic) approach. ii) As for the hit ratio, the online approach is about 6 (resp. 1.3) times better than the random (resp. heuristic) approach. Note that the precision ratio of the offline algorithm is larger than 100%. This is reasonable because the user may repeat watching the same episodes at several time slots and a single prefetching will be regarded as several useful prefetches. iii) Both the precision ratio and the hit ratio of the online algorithm achieve better performance, about 80%,



(a) Precision and hit ratio with different approaches. (b) Improvement compared with random approach.

Fig. 9. Performance comparison in terms of precision and hit ratio.



(a) Costs with different approaches. (b) Cost saving compared with random approach.

Fig. 10. Performance comparison in terms of costs.

whereas the hit ratio of the random approach and the heuristic approach is higher than their precision ratio. This indicates that these two approaches improve the hit ratio with as many prefetchings as possible, which in turn decrease the precision ratio. iv) The performance gap between the heuristic algorithm and the random algorithm indicates the importance of users' behavior-awareness. As shown in Fig. 9(b), compared with the random algorithm, the heuristic algorithm improves both the precision accuracy and the hit ratio by 4 times.

Next, we evaluate the online algorithm in terms of the different costs. We plot the overall cost, the monetary cost, as well as the QoE cost in Fig. 10. As expected, all the costs of the online approach lies between the offline approach and the random approach. The similar monetary cost between the heuristic algorithm and random algorithm indicates the number of prefetchings of these two approaches are comparable, whereas the QoE cost gap is somewhat large, further confirming the requirement of user's watching behavior awareness. In particular, the cost saving for the online approach is slightly smaller than the offline approach. Note that there is a trade-off between the monetary cost and the QoE cost and they depend on the tuning parameter λ_1 . Here, we analyze the performance with the fixed value 0.9. Furthermore, the impact of λ_1 will be analyzed in the following section.

2) *A Case Study*: In order to dive deeper into the detailed performance, we randomly select a user case to study. We plot the number of prefetched content and the distribution of startup delay with different approaches in Fig. 11. We put the server load during all periods when the user watches videos together and plot them in Fig. 11(a). The histograms in Fig. 11(b) is the distribution of startup delay without prefetching. Fig. 11(c)(d) are the corresponding performance with different strategies. From Fig. 11(c), we observe that the random approach decides to prefetch a fixed number of content (e.g., 2 content) regardless of the server load, whereas both

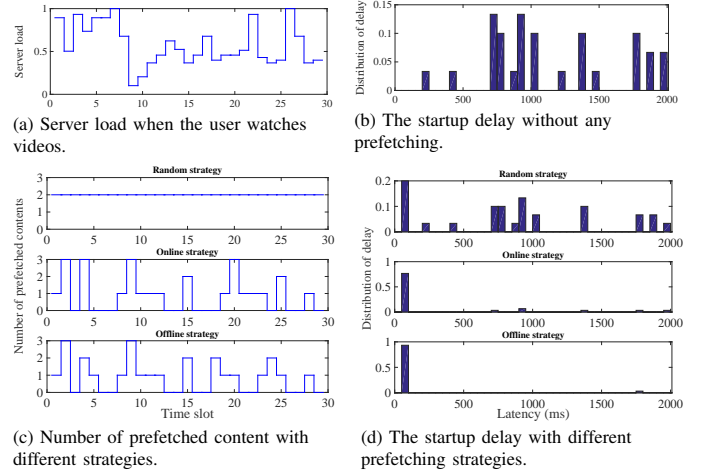


Fig. 11. An example of scheduled content prefetching with different strategies in terms of the number of prefetched content and the startup delay.

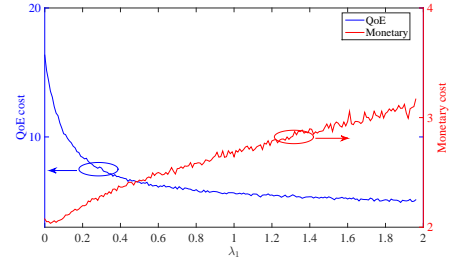


Fig. 12. The impact of trade-off parameter between the monetary-related cost and the QoE-related cost.

the online algorithm and the offline algorithm consider the monetary cost and the QoE cost and strive to avoid prefetching at peak load periods. As for the startup delay presented in Fig. 11(d), we observe that although the random approach prefetches more content (e.g., 60 content), there are more episodes suffering from a large startup delay with the random approach than those of the online and offline approaches. Due to the space limitation, we omit the results of the heuristic algorithm.

3) *Impact of the Trade-off Parameter*: Since the parameter λ_1 balances the trade-off between the monetary-related cost C^m and the QoE-related cost C^q , it is critical to design an intelligent approach which can adaptively schedule the content prefetching in response to different preferences, i.e., if the video service provider pays more attention to clients' perceived experience than the monetary cost, more content should be prefetched; otherwise, content only will be prefetched when the server is idle. We evaluate the adaptive property of our design by quantifying the monetary-related cost C^m and the QoE-related cost C^q with changing parameter λ_1 .

As shown in Fig. 12, we observe that with the increase of λ_1 , the QoE cost decreases. However, as λ_1 increases, the monetary cost also grows, demanding a trade-off between the monetary-related and QoE-related cost in making a better content prefetching decision. The choice of λ_1 depends on the system budget and the desired QoE level.

VI. RELATED WORK

In this section, we survey the related work in the literature, on the edge-network content delivery and the video prefetching.

A. Edge-resource Assisted Content Delivery

Recently, some systems were developed to take advantage of the edge devices to assist the content delivery. Li et al. [21] proposed to utilize the smart APs to realize the offline downloading and claimed that the a proof-of-concept middleware could help users achieve the best expected performance by combining the advantages of cloud-based and AP-based offload downloading. Hu et al. [22] proposed a Voronoi-like partition algorithm to take both the geo-distribution of users' request and Wi-Fi APs into account and conducted the replication in a server peak offloading manner. Ma et al. [23] conducted extensive measurement studies on the content placement strategies of the emerging smart router-based peer video content delivery network in China. Jayasundara et al. [24] proposed to improve the scalability of video-on-demand systems by placing the video content at the end-users' devices. Chen et al. [25] implemented a crowdsourcing-based content distribution system, Thunder Crystal, by utilizing the resources of smart APs. They stimulated users to contribute upload bandwidth by rebating cash.

However, these works just adopted naive content replication policies (e.g., the popularity-based or even random approach). In contrast, our work is able to make optimal content prefetching decisions in an online manner, by learning the users' watching history.

B. Video Prefetching

Prefetching video ahead of users' requests is critical to not only reduce the startup delay but also shift the traffic away during the peak periods. In this part, we classify the previous works based on the different prefetching schemes as follows.

Popularity-based Video Prefetching: Krishnappa et al. [26] studied the Hulu traffic in a campus network and claimed a scheme of prefetching the top-100 popular videos of one week was effective. Liang et al. [27] strived to maximize the byte-hit ratio through selecting the prefetching requests based on the number of users that were about to send the same prefetching request.

Social-aware Video Prefetching: Koch et al. [28] realized the video prefetching by predicting the videos a user may consume from social neighbours. Khemmarat et al. [2] and Cheng et al. [29] investigated the user generated videos in YouTube and presented a prefetching scheme based on the YouTube recommendation system. Wang et al. [30] proposed to reduce startup delay by predicting users' video access patterns based on both the popularity of video and the social closeness in the context of peer-to-peer video on-demand systems. Hu et al. [31] also studied the users' video access patterns in the social community level to reduce the service latency.

User Behavior-aware Video Prefetching: Grigoras et al. [32] modeled the users' interaction with hypermedia documents based on the MDP framework and optimized video content prefetching in anticipation of user-driven navigation. Krishnamoorthi et al. [33] took advantage of parallel TCP connections to do the prefetching with a simple round-robin schedule in the context of interactive branched video streaming. They further [34] proposed to prefetch the alternative videos when the buffer occupancy of the video being viewed reached a threshold.

Our work differentiates from them in several aspects. First, we take the dynamic time-varying server load, which is an important and practical factor, into consideration. Whereas most existing works just ignored this. Second, we systematically propose algorithms to determine the optimal number of prefetched videos, with the objective to achieve a lower accumulated cost and improved QoE. However, none of them gave such an insight. Finally, we utilize the emerging smart AP infrastructures with some prefetching algorithms to efficiently assist the content prefetching.

VII. CONCLUSIONS

In this paper, we propose an AP-assisted content prefetching paradigm to balance the trade-off between the users' QoE and the incurred additional prefetching costs. Our measurement studies on users' AP connection traces and TV series session traces indicate that: i) The user's connected APs are stable over time; ii) More than 60% (*resp.* 90%) users whose over 50% connections are served by their top 1 (*resp.* 5) AP(s); iii) Users tend to watch consecutive episodes in the same TV series. Motivated by these observations, we formulate the content prefetching problem as a Markov Decision Process. Specifically, we obtain the lower and upper performance bound with a random fixed and an offline algorithm, respectively. Moreover, a heuristic algorithm also is proposed as another practical baseline. Finally, we design a reinforcement learning algorithm, which takes both the server load and users' behavior into account, to solve the problem in the online manner. Our trace-driven experiments confirm the superiority of our learning-based approach, which not only achieves a balance between the costs and the users' QoE, but also adapts to the server load.

REFERENCES

- [1] Cisco, "Cisco visual networking index: Forecast and methodology, 2014-2019."
- [2] S. Khemmarat, R. Zhou, D. K. Krishnappa, L. Gao, and M. Zink, "Watching user generated videos with prefetching," in *ACM MMSys*, 2011.
- [3] W. Jiang, S. Ioannidis, L. Massoulié, and F. Picconi, "Orchestrating massively distributed cdns," in *ACM CoNEXT*, 2012.
- [4] iiMedia Research Group, "2015 market report of chinese intelligent router sector."
- [5] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.
- [6] D. S. Berger, P. Gland, S. Singla, and F. Ciucu, "Exact analysis of ttl cache networks: The case of caching policies driven by stopping times," in *ACM SIGMETRICS*, 2014.
- [7] N. C. Fofack, P. Nain, G. Neglia, and D. Towsley, "Analysis of ttl-based cache networks," in *IEEE VALUETOOLS*, 2012.

- [8] Y. Chen, B. Zhang, Y. Liu, and W. Zhu, "Measurement and modeling of video watching time in a large-scale internet video-on-demand system," *IEEE TMM*, 2013.
- [9] A. Carlier, V. Charvillat, and W. T. Ooi, "A video timeline with bookmarks and prefetch state for faster video browsing," in *ACM MM*, 2015.
- [10] T. Mills, "Time series techniques for economists," *Cambridge University Press*, 1990.
- [11] Q. Zheng and B. Veeravalli, "Utilization-based pricing for power management and profit optimization in data centers," *Elsevier JPDC*, 2012.
- [12] Amazon Pricing, <https://aws.amazon.com/pricing/>.
- [13] K.-T. Chen, C.-Y. Huang, P. Huang, and C.-L. Lei, "Quantifying skype user satisfaction," in *ACM SIGCOMM CCR*, 2006.
- [14] R. Bellman, "On the theory of dynamic programming," *PNAS*, 1952.
- [15] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.
- [16] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, 1992.
- [17] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, 1992.
- [18] J. He, Y. Wen, J. Huang, and D. Wu, "On the cost-qoe tradeoff for cloud-based video streaming under amazon ec2's pricing models," *IEEE TCSVT*, 2014.
- [19] S. S. Krishnan and R. K. Sitaraman, "Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs," *IEEE/ACM ToN*, 2013.
- [20] J. Domènech, J. A. Gil, J. Sahuquillo, and A. Pont, "Web prefetching performance metrics: A survey," *Performance Evaluation*, 2006.
- [21] Z. Li, C. Wilson, T. Xu, Y. Liu, Z. Lu, and Y. Wang, "Offline downloading in china: A comparative study," in *ACM IMC*, 2015.
- [22] W. Hu, Z. Wang, M. Ma, and L.-F. Sun, "Edge video cdn: A wi-fi content hotspot solution," *Springer JCST*, vol. 31, no. 6, pp. 1072–1086, 2016.
- [23] M. Ma, Z. Wang, K. Su, and L. Sun, "Understanding content placement strategies in smartrouter-based peer video cdn," in *ACM NOSSDAV*, 2016, p. 7.
- [24] C. Jayasundara, M. Zukerman, T. A. Nirmalathas, E. Wong, and C. Ranaweera, "Improving scalability of vod systems by optimal exploitation of storage and multicast," *IEEE TCSVT*, 2014.
- [25] L. Chen, Y. Zhou, M. Jing, and R. T. Ma, "Thunder crystal: a novel crowdsourcing-based content distribution platform," in *ACM NOSSDAV*, 2015.
- [26] D. K. Krishnappa, S. Khemmarat, L. Gao, and M. Zink, "On the feasibility of prefetching and caching for online tv services: a measurement study on hulu," in *Springer PAM*, 2011.
- [27] K. Liang, J. Hao, R. Zimmermann, and D. K. Yau, "Integrated prefetching and caching for adaptive video streaming over http: an online approach," in *ACM MMSys*, 2015.
- [28] C. Koch and D. Hausheer, "Optimizing mobile prefetching by leveraging usage patterns and social information," in *IEEE ICNP*, 2014.
- [29] X. Cheng and J. Liu, "Nettube: Exploring social networks for peer-to-peer short video sharing," in *IEEE INFOCOM*, 2009.
- [30] Z. Wang, L. Sun, S. Yang, and W. Zhu, "Prefetching strategy in peer-assisted social video streaming," in *ACM MM*, 2011.
- [31] H. Hu, Y. Wen, T.-S. Chua, J. Huang, W. Zhu, and X. Li, "Joint content replication and request routing for social video distribution over cloud cdn: A community clustering method," *IEEE TCSVT*, 2015.
- [32] R. Grigoras, V. Charvillat, and M. Douze, "Optimizing hypervideo navigation using a markov decision process approach," in *ACM MM*, 2002.
- [33] V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri, "Quality-adaptive prefetching for interactive branched video using http-based adaptive streaming," in *ACM MM*, 2014.
- [34] V. Krishnamoorthi *et al.*, "Bandwidth-aware prefetching for proactive multi-video preloading and improved has performance," in *ACM MM*, 2015.